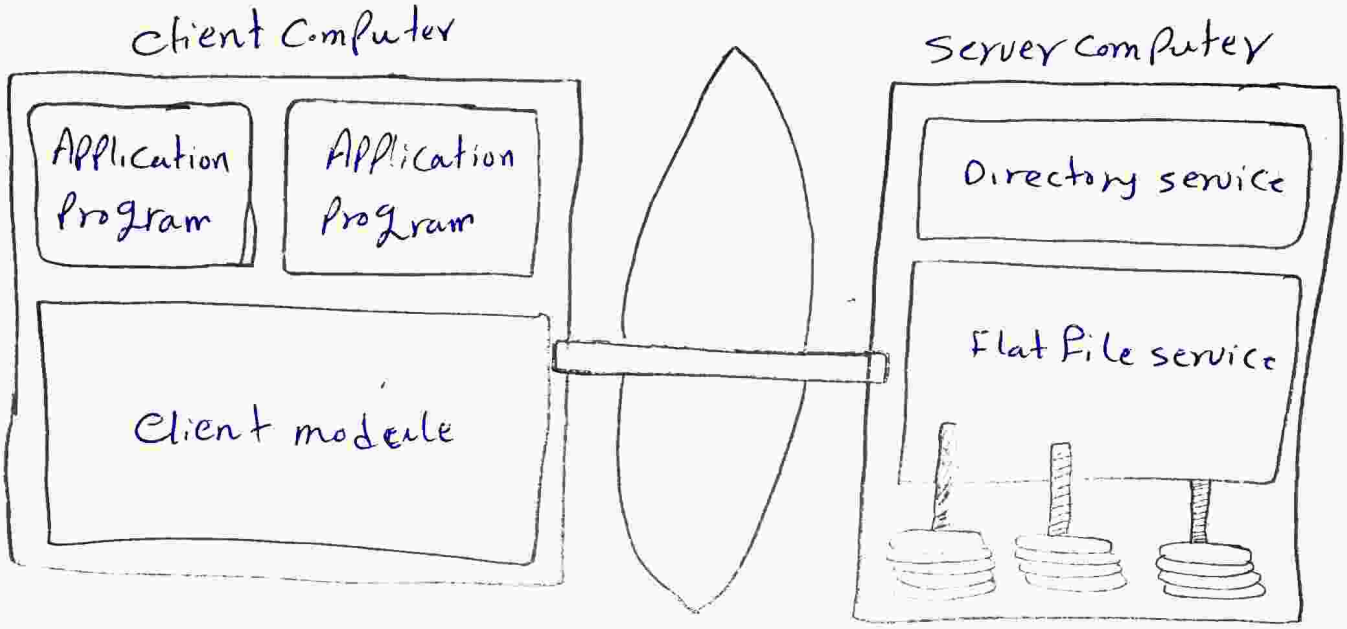
File service architecture ch:3



→ It divides file service to 3 Components

1) Flat file service:-

↳ implements operations on file (create, delete, read, write, access control operations, ...)

↳ uses UFID to refer to all files in all requests.

2) Directory service:-

→ Provides mapping between text names and UFID for each file for flat file service.

→ creates and updates directories (hierarchical file structures)

3) client service:-

→ client of directory and flat file service.

→ clients hold information about the location of flat file server and directory server processes.

→ It runs in each client's computer ⇒ integrating and expanding flat file and directory services to provide unified (API)

file service interface

True (flat) file service

→ operations on individual files

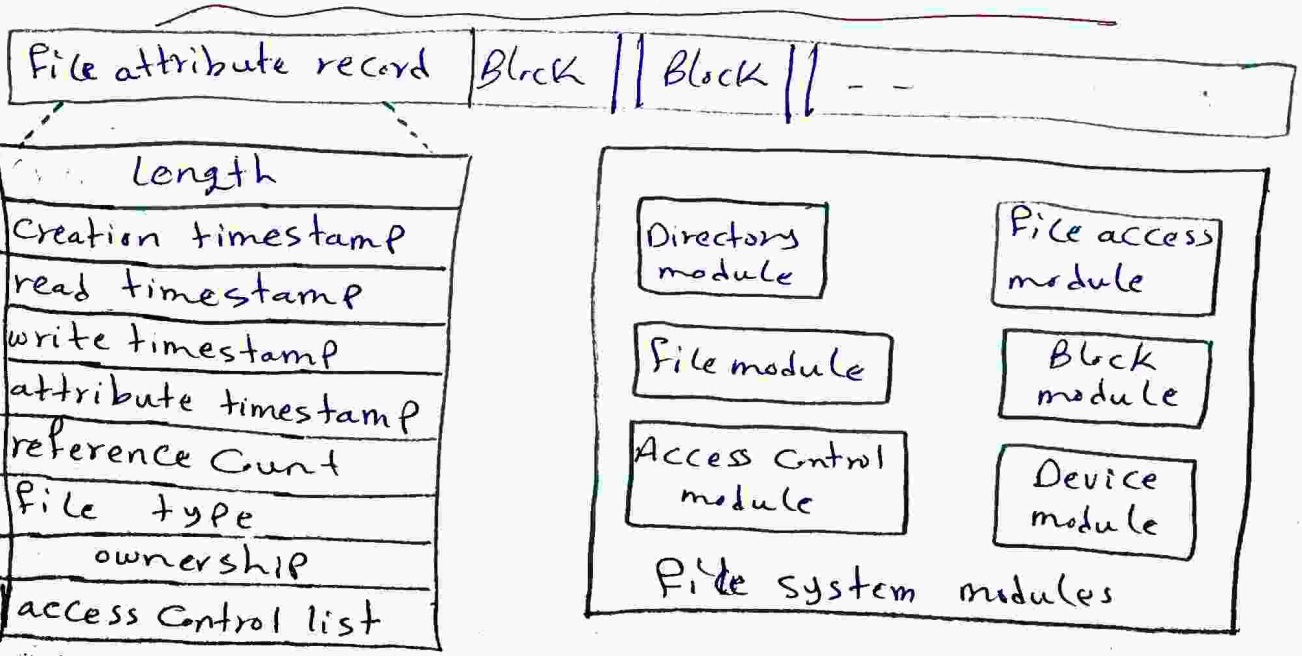
→ reading, writing

directory service

→ create & manage directories

→ adding files from directories.

File Attributes & system modules



File system modules

* Directory modules: relates file names to file IDs.
* File module :- relates file IDs to particular files.
* Access control module : checks permission for operation requested.
* File Access module : reads or writes file data or attributes.
* Block module: access & allocate disk blocks.
* Device module:- disk I/O and buffering.

⇓

for single host file system.

Access Control

في ال (UNIX) ال (access rights) بتاعة ال (user) بيتعمل ليها (check) مقابل ال (access mode requisted in open call)

وال (file) بيكون open بس لو ال (user) عنده access rights ضرورية.

Note → in DFS, user identity has to be passed with requestes.

في البداية الـ user لازم يعمل (authenticate) مع (server)

ثم يحصل على (check) في الوقت اللي فيه هيتحول اسم الـ file لـ UFID (uniqueFileId) والنتيجة هيتعمل ليها (encoding) وعشان (client) يقدر يستخدم الـ (capability) اللي بدورها بترجع للـ (client) أي عملية "Access" قادرة.

* capability

~~(access) (objects)~~

↳ list of objects allowed to access and type of access allowed [could be broken up per (user, obj)]

— مع كل (client request) الـ (user identity) يتبعتلها (submit) ويحصل (access check) لكل عملية تحاول الـ (files)

File services models

1) Upload/download model (Cached System)

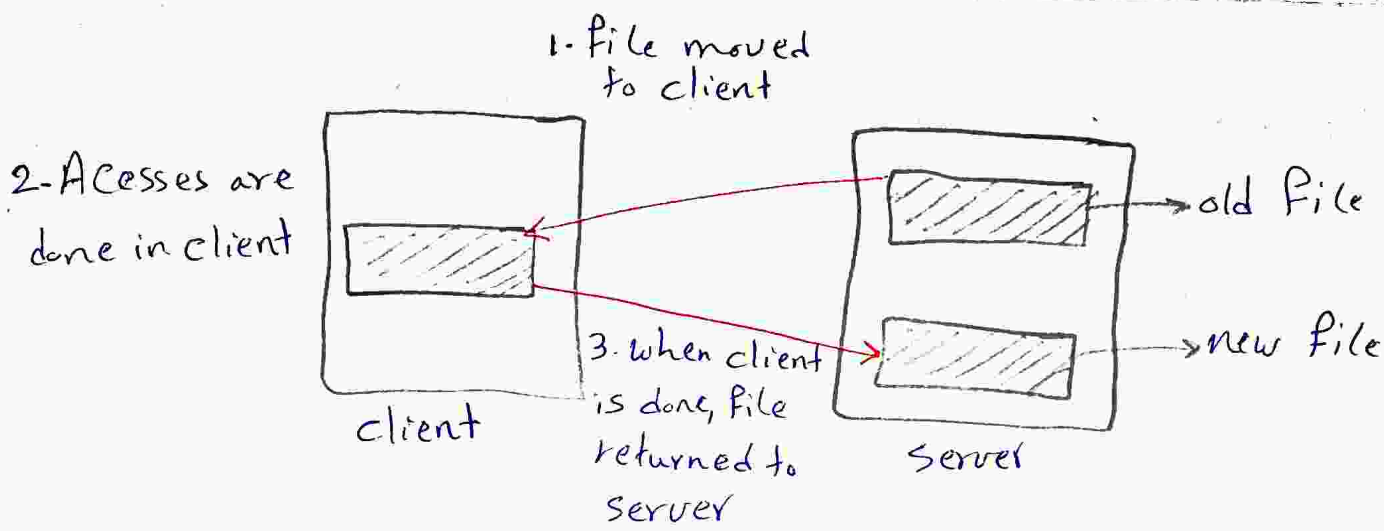
— الـ (file service) بتوفر عمليتين readFile / writeFile

a) ~~or~~ read file

→ transfer entire file from server to client

→ file identified with one master copy existing at server machine but copies of file scattered (متبعثرة) in different caches.

→ reduce network traffic by retaining recently accessed disk blocks in a cache ⇒ repeated accesses to same information can be handled locally.



b) write file operation → transfer entire file in other direction from client to server.

Advantages remote accessed handled by local cache.

↳ most served as fast as local ones.

↳ servers contacted only occasionally

- ↳ reduces server load and network traffic.
- ↳ enhances potential for scalability.

→ require local usages of files so it is simple.

Disadvantages

لما ال (storage) بتاعة ال (client) صغيرة (لن) تتحمل
ال (file) وحتى لو ال (client) بيستخدم جزء من الملف ،
الملف هيعمل (transfer) لل BW network , wasting time .

cache-consistency problem

↳ keep cached copies consistent with master file.

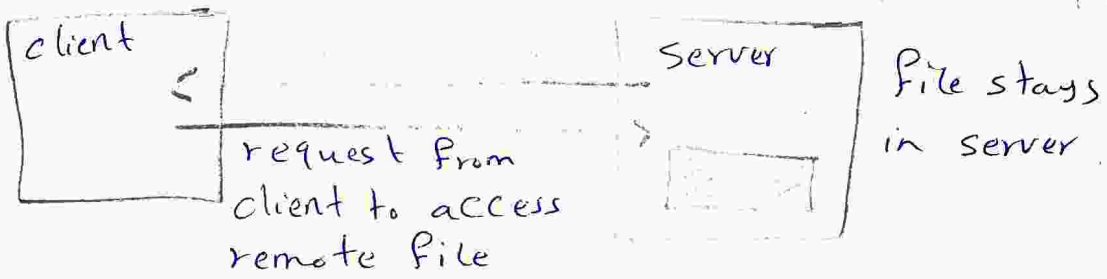
↳ could be called network virtual memory.

[2] Remote access model (Remote service)

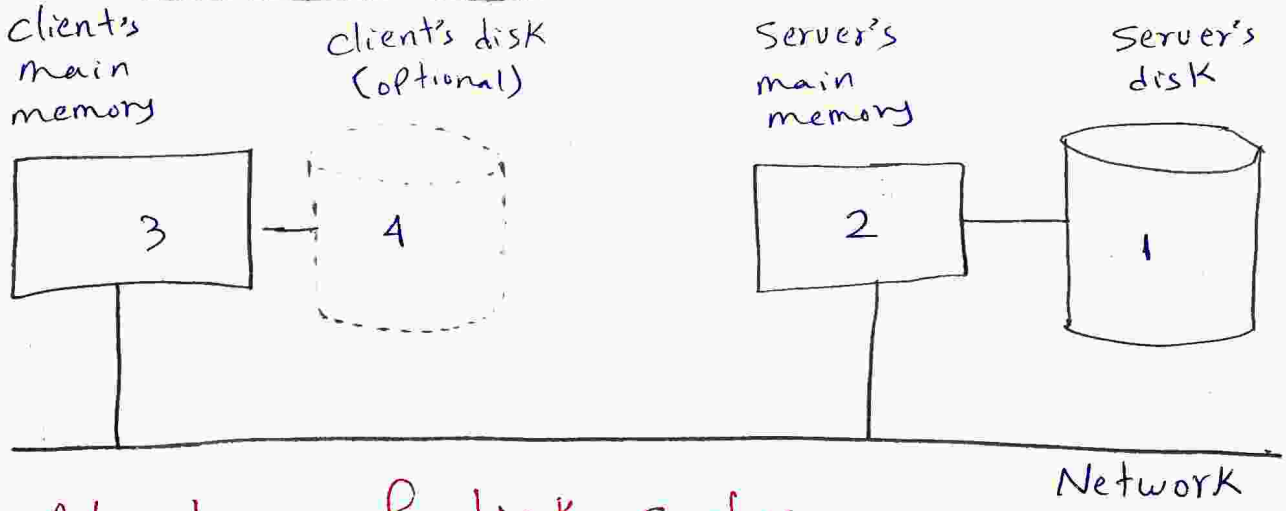
→ server implements all file actions.

→ file service provides many operation for (open, close files & read/write parts of file & move inside file seeking something, ...etc)

→ file system run on servers not clients.



Cache location - Disk vs. Main memory



Advantages of disk caches :-

- more reliable.
- cached data (which kept on disk) still there during recovery & no need to be fetched again.

Advantages of main-memory caches

↳ permit workstations to be diskless

↳ data can be accessed more quickly.

↳ more performance speed in bigger memories.

↳ permits single caching mechanism for servers and users (cause they are in main memory)

Consistency

(master copy) ال مقارنة مع (locally cached copy) ال هل لـ

* client-initiated approach
  1) client initiates a validity check.
  2) server checks whether local data are consistent with master copy.
* server-initiated approach:-
  1) server records "for each client" the parts of files it caches.
  2) when server detects potential inconsistency, it must react.

Cache-update Policy

* Write through:

↳ write data through to disk as soon as they are placed on any cache.

↳ When another client reads file, He'll get the most update from server

⇒ reliable but poor performance.

* Delay-write

↳ modifications written to the cache and then written through to server later.

⇝ write accesses complete quickly (some data may be overwritten before they are written back, so need to be written)

⇝ Poor reliability: unwritten data will be lost whenever a user machines Craches.

Comparing Caching & Remote service

المقارنة في جدول في آخر صفحة

* In Caching:

→ many remote accesses handled efficiently by local cache.

→ most " " will be served as fast as local ones.

⇝ servers are contacted only occasionally

↳ reduces server load & network traffic.

↳ Enhances potential for scalability.

*Remote server

→ method handles every remote access accross the network.

→ Penalty in network traffic, server load and performance.

→

* Total network overhead in transmitting big chunks of data (Caching) is lower than series of responses to specific requests (remove service)

* Caching is superior in access patterns with infrequent writes.

* for execution on machines with local disks or large main memories, Caching does well.

* for diskless & small-memory capacity remote service is better.

* In Caching, lower intermachine interface and upper user interface are different.

* In Remote service, intermachine interface mirrors local user-file system interface.

Note ممكن نستبدل الجدول بالوحدة دي

وآخر جزء في الوحدة السابقة.

State of service and clients

~~(Service) ... (state) ...~~

(clients) لا (maintain) بـ (service) لا ـ

(stateful & stateless) على حالتين

1) stateless file server :-

→When client sends a request to server ⇒ server carries request ⇒ sends the reply ⇒ remove from its internal tables all information about request.

⇒ there is no client - information kept on server, between requests.

↳ This means it avoids state information by making each request self-contained (يتعامل لوحده) to help server to do its work.

↳ No need to establish and terminate connection by open & close operations.

↳ Poor support for locking or synchronization among concurrent accesses.

Stateful file service

→ Server has information about client between requests

→ When client opens a file ⇒ server fetches information about file from its disk ⇒ stores it in its memory

⇒ Gives the client a unique connection identifier for the open file.

↳ server has info. about which client has which file open.

↳ identifier used for subsequent accesses.

↳ server must reclaim main memory space used by clients who are no longer active.

* Increased Performance

↳ fewer disk accesses.

DFS-server symantics Comparison

→ Falirue recovery:

In stateful server a) it loses all volatile state in crash.

b) restore old state about clients.

c) server needs to be aware of crashed client processes.

In stateless server

↳ failure & recovery are almost un-noticeable.

Comparison

	Caching	remote-Service
Handling remote accesses	↳most of them handled efficiently by local cache.	It handles every remote access accross the network
server load & network traffic	reduced	↳un-available
Total network overhead (in transmitting)	lower	higher
access Patterns with infrequent writes	more successiful	less successiful
execution in local disks or large main-memories	better used	lower
for diskless & small memories	less in benefit	better used
relation between lower intermachine interface & user interface	different	one of them mirrors the other one.